课程回顾

- 近似算法应用例子：
  - 顶点覆盖问题
  - 旅行商问题（满足三角不等式存在常数近似比算法）
  - 集合覆盖问题
  - 子集和问题
- 回溯法 vs. 分支限界法
  - 找出满足约束条件的所有解 vs. 找出满足约束条件某种意义下的一个解
  - 深度优先搜索 vs. 广度优先搜索
  - 子集树与排列树

第8章 随机算法

苏州大学 计算机科学与技术学院

汪笑宇

Email: xywang21@suda.edu.cn

本章内容

- 随机化算法（参考书目Chapter 7）
- 概率分析和随机算法（教材Chapter 5）

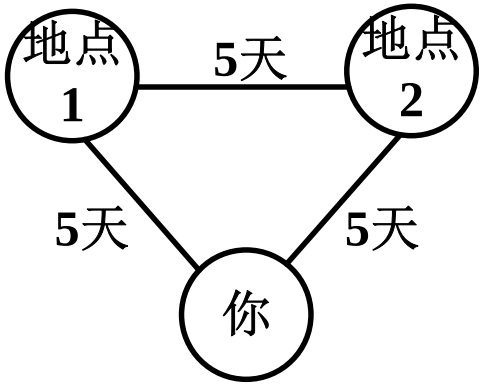
本章目录

- 随机算法概述
- 概率分析相关知识
- 数值随机化算法
- Sherwood算法
- Las Vegas算法
- Monte Carlo算法

本章目录

引入

- 想象自己是神话故事的主人公，有一张不易懂的地图，上面描述了一处宝藏的藏宝地点。经分析你能确定最有可能的两个地点是藏宝地点，但二者相距甚远
- 假设你如果已到达其中一处，就立即知道该处是否为藏宝地点
- 你到达两处之一地点，以及从其中一处到另一处的距离是5天的行程
- 进一步假设有一条恶龙，每晚光顾宝藏并从中拿走一部分财宝
- 假设你取宝藏的方案有两种：
  - 方案1：花4天的时间计算出准确的藏宝地点，然后出发寻宝，一旦出发不能重新计算
  - 方案2：有一个小精灵告诉你地图的秘密，但你必须付给他报酬，相当于恶龙三晚拿走的财宝



假设宝藏价值x
恶龙一天取走y
方案1：x-9y
方案2：x-8y

引入 (续)

- 是否有其他方案？
  - 方案3：投硬币决定先到一处，失败后到另一处（冒险方案）
    - 一次成功所得：x-5y， 机会1/2
    - 二次成功所得：x-10y，机会1/2
    - 期望盈利：x-7.5y
- 当一个算法面临某种选择时，有时随机选择比耗时做最优选择更好，尤其是当最优选择所花的时间大于随机选择的平均时间的时候

随机算法只是期望的时间更有效，
但有可能遭受到最坏的可能性

随机算法概述

- 随机化算法允许算法在执行过程中随机地选择下一个计算步骤
- 在许多情况下， 当算法在执行过程中面临一个选择时， 随机性选择常比最优选择省时，因此，随机化算法可在很大程度上降低算法的复杂度
- 随机化算法的一个基本特征是对所求解问题的同一实例用同一随机化算法求解两次可能得到完全不同的效果，这两次求解所需的时间甚至得到的结果可能有相当大的差别

期望时间 vs. 平均时间

- 确定性算法的平均执行时间
  - 输入规模一定的所有输入实例是等概率出现时，算法的平均执行时间
- 随机算法的期望执行时间
  - 反复解同一个输入实例所花的平均执行时间
- 随机算法可讨论如下两种期望时间：
  - 平均的期望时间：所有输入实例上平均的期望执行时间
  - 最坏的期望时间：最坏的输入实例上的期望执行时间

随机算法特点

- 不可再现性
  - 在同一个输入实例上，每次执行结果不尽相同
    - n皇后问题：随机算法运行不同次将会找到不同的正确解
    - 找一给定合数的非平凡因子：每次运行的结果不尽相同，但确定性算法每次运行结果必定相同
- 分析困难
  - 要求有概率论、统计学和数论的知识

随机算法分类

■随机算法分类：

- 数值随机化算法常用于数值问题的求解，得到的往往是近似解， 且近似解的精度随计算时间的增加而不断提高
- 蒙特卡罗算法用于求问题的准确解，例如判定问题，其解为"是"或"否"， 二者必居其一，不存在任何近似解答。用蒙特卡罗算法能求得问题的一个解，但这个解未必是正确的
- 拉斯维加斯算法不会得到不正确的解。一旦用拉斯维加斯算法找到一个解，这个解就一定是正确解， 但有时找不到解
- 舍伍德算法总能求得问题的一个解，且求得的解总是正确的。当一个确定性算法在最坏情况下的计算复杂性与其在平均情况下的计算复杂性有较大差别时，可在这个确定性算法中引入随机性将它改造成一个舍伍德算法，消除或减少问题的好坏实例间的这种差别

本章目录

- 随机算法概述
- 概率分析相关知识
- 数值随机化算法
- Sherwood算法
- Las Vegas算法
- Monte Carlo算法

生活中的概率问题——购买盲盒

■购买盲盒

- 一套一共b款
- 每款买到的概率是$1/b$（假设商家很良心）

平均意义上需要买多少个盲盒才能集齐全套呢？

生活中的概率问题——购买盲盒 (续)

- 假设我们一次购买买到一个之前没买到的新的款式，则称作一次命中（hit）
  问题转化为：求有b次命中时，购买次数n的期望
- 将购买次数n分成若干个阶段，第i阶段表示从第$i-1$次命中到第i次命中之间的购买，购买次数为n_i，服从几何分布
- 第i阶段得到一次命中的概率：
  未买到的款式个数/总款式个数 $= (b - i + 1)/b$

几何分布：一系列伯努利试验，其中每次成功概率为p、失败概率为$1-p$，在获得一次成功前要进行的试验次数服从几何分布
几何分布期望为$1/p$

生活中的概率问题——购买盲盒 (续)

■第i阶段：

已有

购买1
重复

购买2
重复

购买3
重复

购买4
重复

购买5
重复

…

购买n_i
新款！

➢n_i服从$p = (b - i + 1)/b$的几何分布

生活中的概率问题——购买盲盒 (续)

■第i阶段购买次数为n_i，则得到b次命中的购买次数$n = \sum_{i=1}^{b} n_i$

$$E[n_i] = \frac{b}{b-i+1}$$

$$E[n] = E\left[\sum_{i=1}^{b} n_i\right] = \sum_{i=1}^{b} E[n_i] = \sum_{i=1}^{b} \frac{b}{b-i+1}$$

$$= b\sum_{i=1}^{b} \frac{1}{i} = b(\ln b + O(1))$$

大概需要购买$b\ln b$次才能集齐所有盲盒
（前提是各款式等概率出现）
假设b=30，则购买次数≈103

生活中的概率问题——生日悖论

- 一个屋子里人数必须达到多少人，可以期望其中有两个人生日相同？
- 假设：
  - 所有年份都是n天，不考虑闰年
  - 所有人生日都是均匀分布在一年的n天中的，即$\Pr\{b_i=r\} = 1/n$，其中b_i表示编号为i的人的生日，r表示一年n天中的一天
  - 每两个人生日都是独立的

生活中的概率问题——生日悖论 (续)

- 两个人i和j生日正好都在r这一天的概率为：

$$\Pr\{b_i=r \text{ and } b_j=r\} = \Pr\{b_i=r\}\Pr\{b_j=r\} = 1/n^2$$

- 他们的生日刚好在同一天的概率：

$$\Pr\{b_i = b_j\} = \sum_{r=1}^{n} \Pr\{b_i = r \text{ and } b_j = r\} = \sum_{r=1}^{n} (1/n^2) = 1/n$$

- 设置指示变量X_{ij}表示两个人i和$j(i\neq j)$的生日情况：

$$X_{ij} = \begin{cases} 1, & \text{if person } i \text{ and person } j \text{ have the same birthday,} \\ 0, & \text{otherwise.} \end{cases}$$

$$E[X_{ij}] = 1 \cdot \Pr\{X_{ij} = 1\} + 0 \cdot \Pr\{X_{ij} = 0\} = \Pr\{b_i = b_j\} = 1/n$$

生活中的概率问题——生日悖论 (续)

■设X表示计数生日相同两人对数目的随机变量，且屋子里一共有k人，则有

$$X = \sum_{i=1}^{k} \sum_{j=i+1}^{k} X_{ij}$$

■等号两边取期望得

$$E[X] = E\left[\sum_{i=1}^{k} \sum_{j=i+1}^{k} X_{ij}\right] = \sum_{i=1}^{k} \sum_{j=i+1}^{k} E[X_{ij}]$$

$$= C_k^2 \cdot \frac{1}{n} = \frac{k(k-1)}{2n} \geq 1$$

解得 $k \geq \sqrt{2n} + 1$

当$n = 365$时，$k \geq 28$

生活中的概率问题——抽奖

- 设抽奖箱中有不少于a $(0 < a < 1)$比例的一等奖，有放回的抽奖多少次可以保证抽到一等奖的概率不小于b $(0 < b < 1)$？

- 对立事件：
  至少抽到一次一等奖 vs. 抽到的全部不是一等奖

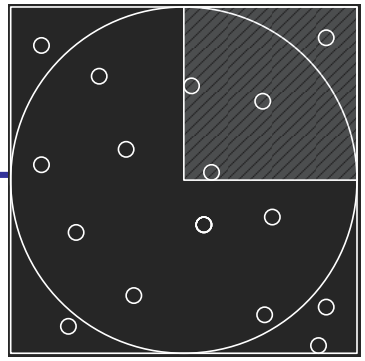
- 假设抽奖k次，至少抽到一次一等奖的概率
  $\geq 1 - (1 - a)^k \geq b$
  即$k \geq \log_{1-a}(1 - b)$

假设a=1%, b=0.9，则k≈230

本章目录

- 随机算法概述
- 概率分析相关知识
- **数值随机化算法**
- Sherwood算法
- Las Vegas算法
- Monte Carlo算法

数值随机化算法



- π值计算
  - 实验：将n只飞镖随机投向一正方形的靶子，计算落入此正方形的内切圆中的飞镖数目k
  - 假定飞镖击中方形靶子任一点的概率相等（用计算机模拟比任一飞镖高手更能保证此假设成立）
- 设圆的半径为r，圆面积$S_1 = \pi r^2$；方靶面积$S_2 = 4r^2$
- 由等概率假设可知落入圆中的飞镖和正方形内的飞镖平均比为：

$$\frac{k}{n} = \frac{\pi r^2}{4r^2} = \frac{\pi}{4}$$

- 由此可知：$\pi \approx \frac{4k}{n}$

数值随机化算法 (续)

■求π近似值的算法（仅以右上1/4区域为样本）

```
DARTS(n)
1   k ← 0
2   for i ← 1 to n do
3       x ← RANDOM(0, 1)
4       y ← RANDOM(0, 1) // 随机产生点(x, y)
5       if x^2 + y^2 ≤ 1
6           k ← k + 1
7    return 4k/n
```

■实验结果展示

- n = 1000万：3.140740, 3.142568 (2位精确)
- n = 1亿：3.141691, 3.141363 (3位精确)
- n = 10亿：3.141527, 3.141507 (4位精确)

本章目录

- 随机算法概述
- 概率分析相关知识
- 数值随机化算法
- Sherwood算法
- Las Vegas算法
- Monte Carlo算法

雇用问题

- 假如你要雇用一名新的办公助理，雇用代理每天给你推荐一个应聘者，你面试这个人，然后决定是否雇用他
- 支出：支付给雇用代理费用c_i以面试应聘者、支付给雇用者雇用费用c_h

```
HIRE_ASSISTANT(n)
1   best ← 0
2   for i ← 1 to n do
3       interview candidate i
4       if candidate i is better than candidate best
5           best ← i
6           hire candiate i
```

总费用：$O(c_i n+c_h m)$

m：雇用人数

面试者之间相互商量？

雇用问题 (续)

■改为随机算法：

```
RANDOMIZED_HIRE_ASSISTANT(n)
1   randomly permute the list of candidates
2   best ← 0
3   for i ← 1 to n do
4       interview candidate i
5       if candidate i is better than candidate best
6           best ← i
7           hire candiate i
```

雇用费用期望：$O(c_h \ln n)$

Sherwood算法

Sherwood算法

- 分析确定性算法在平均情况下的时间复杂度时，通常假定算法的输入实例满足某一特定的概率分布
- 很多算法对于不同输入实例运行时间差别很大，可采用Sherwood算法消除时间复杂度与输入实例间的依赖关系
- 通常有两种方式：
  - 在确定性算法的某些步骤引入随机因素
  - 仅对输入实例随机处理，再执行确定性算法

Sherwood算法 (续)

■例：快速排序

- ➢平均时间复杂度$O(n\lg n)$
- ➢有序数列排序：时间复杂度$O(n^2)$

初始序列：30 25 19 12 6 ⟶ 一次划分：[6 25 19 12] 30 []

划分不均衡

- ➢如何提升最坏情况性能？
  - 想法一：每次随机选择划分元 (pivot)

初始序列：30 25 19 12 6 ➡ 位置调整：19 25 30 12 6 ➡ 一次划分：[6 12] 19 [25 30]

随机选择

划分较均衡

  - 想法二：把初始序列打乱

初始序列：30 25 19 12 6 ➡ 打乱顺序：12 30 6 19 25 ➡ 一次划分：[6] 12 [30 19 25]

划分较均衡

Sherwood算法的执行时间

■Sherwood算法能够平滑不同输入实例的执行时间

A：一个确定性算法

$t_A(x)$：用算法A解实例x的执行时间

X_n：大小为n的输入实例集合

X_n中每一个实例是等可能出现的

→ 平均执行时间：

$$\bar{t}_A(n) = \sum_{x \in X_n} t_A(x)/|X_n|$$

? 存在$x \in X_n$,使得：

$t_A(x) >> \bar{t}_A(n)$

→

B：一个随机算法（Sherwood算法）

$s(n)$：算法B为取得均匀性所付出的成本

对于任意$x \in X_n$执行时间期望值：$t_B(x) \approx \bar{t}_A(n) + s(n)$

Sherwood算法的执行时间 (续)

对于任意 $x \in X_n$ 执行时间期望值：$t_B(x) \approx \bar{t}_A(n) + s(n)$

平均期望时间：

$$\bar{t}_B(n) = \sum_{x \in X_n} t_B(x)/|X_n| \approx \bar{t}_A(n) + s(n)$$

- Sherwood算法的平均执行时间略为增加
- 不再有最坏情况的实例，但有最坏的执行时间

Sherwood算法实例——快速排序

```
RAND_QUICKSORT(A, low, high)
1   //A: 待排序数组，low/high: 排序起始/终止下标
2   if low < high
3       i ← RANDOM(low, high); //low..high随机抽取一个下标
4       swap(A[low], A[i])
5       k ← PARTITION(A, low, high)
6       RAND_QUICKSORT(A, low, k-1)
7       RAND_QUICKSORT(A, k+1, high)
```

引入随机因素

```
SHUFFLE(A)
1   n ← A.length
2   for i ← 1 to n-1 do
3       //在A[i..n]中随机选一个元素放在A[i]上
4       j ← RANDOM(i, n)
5       swap(A[i], A[j])
6   执行原确定性算法
```

原算法较复杂，很难对其进行修改时可适用

输入实例随机处理

Sherwood算法应用——随机的预处理

$f : X \to Y$：解某问题用到的函数，且相应算法平均性能较优

X_n：大小为n的输入实例集合

A_n：和X_n大小相同的集合，A_n中能够有效地均匀随机抽样

$A = \cup A_n, \quad X = \cup X_n$

$u : X \times A \to X, \quad v : A \times Y \to Y$

原实例x可通过随机抽样变换成另一个实例z

对z的解可变换为对原实例x的解

函数u和v在最坏情况下能够有效计算

Sherwood算法应用——随机的预处理 (续)

■$f(x)$对应的确定性算法可改造为Sherwood算法

```
RH(x)                          //用Sherwood算法计算f(x)
1   n ← x.length               //x的大小为n
2   r ← RANDOM(A_n)            //随机取一元素
3   z ← u(x, r)                //将原实例x转化为随机实例z
4   s ← f(z)                   //用确定性算法求z的解s
5   return v(r, s)             //将解s变换为x的解
```

Sherwood算法应用——随机的预处理 (续)

■随机的预处理提供了一种加密计算的可能性

- 想针对某个实例x计算$f(x)$
- 缺乏计算能力或有效算法
- 别人可提供服务计算
- 不想泄露输入实例x

1. 使用函数u将x加密为某一随机实例z
2. 将z提交给f计算出$f(z)$的值
3. 使用函数v转换为$f(x)$

本章目录

- 随机算法概述
- 概率分析相关知识
- 数值随机化算法
- Sherwood算法
- Las Vegas算法
- Monte Carlo算法

Las Vegas算法与Monte Carlo算法

■给定n个元素（n非常大）的无序序列A，已知至少有一半元素大于k，我们想找到任一个元素值大于k的序列下标

```
repeat
    j ← RANDOM(1, n)
until A[j] > k
return j
```

```
for i ← 1 to m do   // m远小于n
    j ← RANDOM(1, n)
    if A[j] > k
        return j
return 0  // 0表示未找到
```

赌时间而不赌正确性

要么一定返回正确解，要么无解

Las Vegas算法

赌正确性而不赌时间

一定返回解，但可能不正确

Monte Carlo算法

Las Vegas算法

■特点

- ➢要么返回正确的解，要么随机决策导致一个僵局
- ➢若陷入僵局，使用同一实例运行同一算法，有独立的机会求出解
- ➢成功的概率随着执行时间的增加而增加

■算法的一般形式

```
OBSTINATE(x)
1   repeat
2       LV(x, y, success)
3   until success
4   return y
```

x：输入实例，y：返回值
success：布尔值指示执行成功/失败

Las Vegas算法 (续)

■设$t(x)$是算法OBSTINATE找到一个正确解的期望时间，则

```
OBSTINATE(x)
1   repeat
2       LV(x, y, success)
3   until success
4   return y
```

正确的算法：$\forall x,\ p(x) > 0$

更好的情况：$\forall x, \exists \delta > 0,\ p(x) > \delta$

- $s(x)$：对于实例x，单次LV算法成功时的期望时间
- $e(x)$：对于实例x，单次LV算法失败时的期望时间

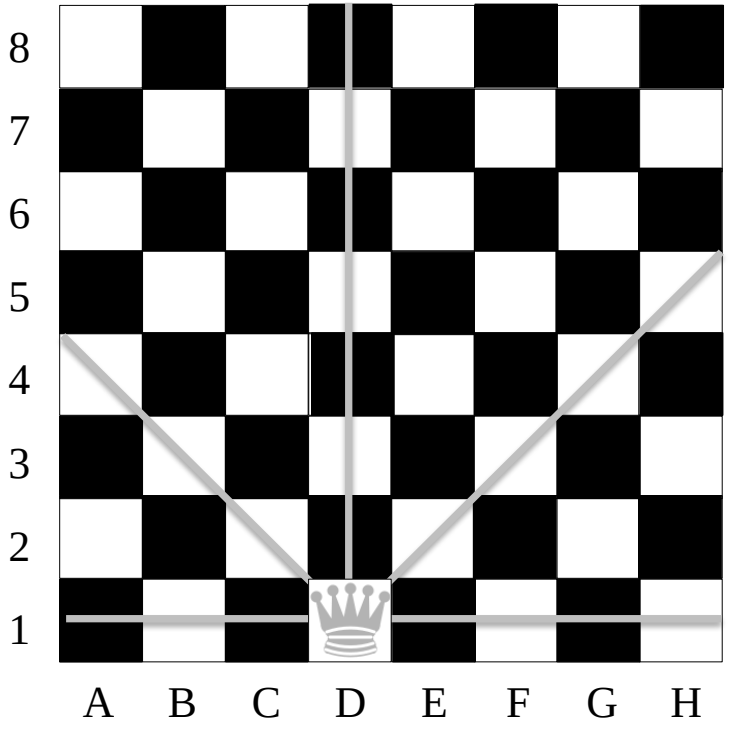
$$t(x) = \underbrace{p(x)s(x)}_{\text{LV算法成功}} + \underbrace{(1-p(x))(e(x)+t(x))}_{\text{LV算法失败}}$$

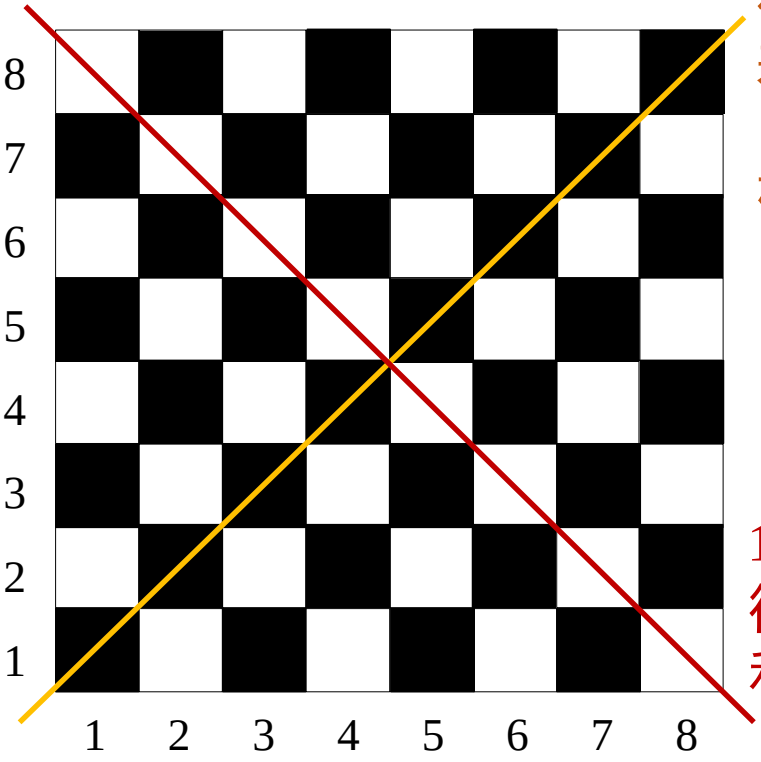
$$\Rightarrow \quad t(x) = s(x) + \frac{1-p(x)}{p(x)} e(x)$$

若要最小化 $t(x)$，则需在$p(x)$, $s(x)$和$e(x)$之间进行某种折衷

Las Vegas算法应用——八皇后问题

- 问题：在8×8棋盘上摆放8个皇后，使其不能互相攻击，即任意两个皇后都不能处于同一行、同一列或同一斜线上



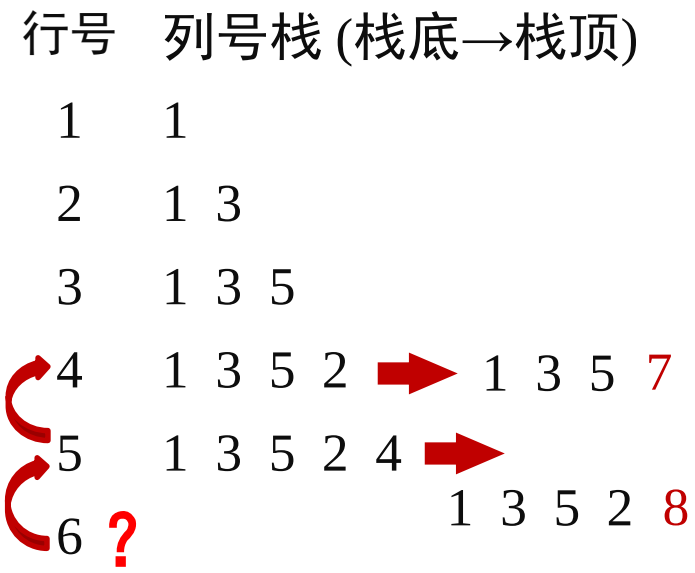


45°斜线：行列号之差相等（*i*-*j*或*j*-*i*相等）

135°斜线：行列号之和相等（*i*+*j*相等）

Las Vegas算法应用——八皇后问题 (续)

```
QUEENS_BACKTRACK()
1  i ← 1,  j ← 1
2  while i ≤ 8 do                  //当前行号i≤ 8
3      从当前列 j 起向后逐列试探，寻找安全列号
4      if 找到安全列号
5          将列号 j 入栈            //(i, j)位置放置皇后
6          i ← i + 1                //将下一行置为当前行
7          j ← 1                    //当前列置为1
8      else
9          i ← i – 1                          //回溯到上一行
10         j 赋值为栈顶值，并退栈  //移除当前皇后
11         j ← j + 1                          //下一列作为当前列
```



Las Vegas算法应用——八皇后问题 (续)

■向量*try*[1..8]中存放结果

- ➢*try*[*i*]——表示第*i*个皇后放在(*i*, *try*[*i*])位置上
- ➢*try*[1..*k*]为*k*-promising：
- ➢对任意 $i, j \in \{1, 2, ..., k\}$，若$i \neq j$，则
  - 无行冲突：第*i*个皇后放在第*i*行
  - 无列冲突：对任意不同的两行*i*, *j*，其列号之差不为0
  - 斜线无冲突：
    135°斜线：不会产生 $i + try[i] = j + try[j]$ 即 $try[i] - try[j] = j - i$
    45°斜线：不会产生 $i - try[i] = j - try[j]$ 即 $try[i] - try[j] = i - j$

对于八皇后问题，解是8-promising的

Las Vegas算法应用——八皇后问题 (续)

```
QUEENS_LV(success) //若success=true，则try[1..8]包含8后问题的一个解
1   col, diag45, diag135 ← ∅      //冲突列及两斜线集合初始为空
2   k ← 0     //行号
3   repeat    //try[1..k]是k-promising，考虑放第k+1个皇后
4       count ← 0   //计数器，count值为第k+1个皇后的安全位置总数
5       for i ← 1 to 8 do    //i是列号，试探(k+1, i)是否安全
6           if i ∉ col and i – k – 1 ∉ diag45 and i + k + 1 ∉ diag135
7               count ← count + 1
8               if RANDOM(1, count) = 1 //以1/count概率在count个安全位置上
9                   j ← i                //随机选择1个位置j放置皇后
10      if count > 0          //count=0时无安全位置，第k个皇后尚未放好
11          k ← k + 1         //try[1..k+1]是(k+1)-promising
12          try[k] ← j
13          col ← col ∪ {j}
14          diag45 ← diag45 ∪ {j – k}
15          diag135 ← diag135 ∪ {j + k}
16  until count = 0 or k = 8   //当前皇后位置搜索失败或8-promising时结束
17  success ← (count > 0)
```

Las Vegas算法应用——八皇后问题 (续)

■算法分析

- p： QUEENS_LV算法一次成功的概率
- s：成功时搜索的结点的平均数　s=9 (空向量try算在内)
- e：失败时搜索的结点的平均数
- p和e理论上难计算，用计算机实验可计算出：
  - $p = 0.1293\ldots$
  - $e = 6.971\ldots$
- 重复上述算法，直至成功时所搜索的平均结点数：

$$t = s + \frac{1-p}{p}e = 55.927...$$

大大优于回溯法，回溯法约为114个结点才能求出一个解

Las Vegas算法应用——八皇后问题 (续)

■算法存在的问题及改进

- ➢消极：LV算法过于消极，一旦失败，从头再来
- ➢乐观：回溯法过于乐观，一旦放置某个皇后失败，就进行系统回退一步的策略，而这一步往往不一定有效
- ➢折中：
  - 先用LV方法随机地放置前k个皇后
  - 然后使用回溯法放置后$(8-k)$个皇后，但不考虑重放前k个结点
    - 若前面的随机选择位置不好，可能使得后面的位置不成功
    - 随机放置的皇后越多，后续回溯阶段的平均时间就越少，失败的概率也越大

一半略少的皇后随机放置较好

Las Vegas算法应用——八皇后问题 (续)

■改进算法

```
16  until count = 0 or k = 8   //当前皇后位置搜索失败或8-promising时结束
17  success ← (count > 0)
```

```
16  until count = 0 or k = stepVegas
17  if count > 0          //已随机放好stepVegas个皇后
18      QUEENS_BACKTRACK(k, col, diag45, diag135, success)
19  else success ← false
```

Monte Carlo算法

■特点

- 偶尔会出错，但对任何实例均能以高概率找到正确解
- 算法运行次数越多，得到正确解的概率越高

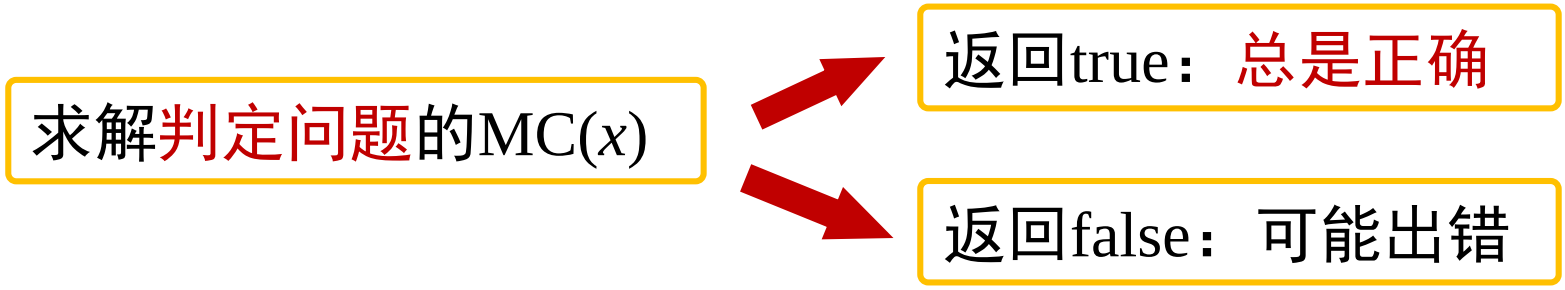
■基本概念

- 设p是一个实数，且$1/2 < p < 1$，若一个Monte Carlo算法以不小于p的概率返回一个正确的解，则该MC算法称为 p 正确，算法的优势 (advantage) 是 $p - 1/2$
- 若一个Monte Carlo算法对同一实例决不给出两个不同的正确解，则该算法称是相容的 (consistent) 或一致的

为了增加一个一致的、p 正确算法成功的概率，只需多次调用同一算法，然后选择出现次数最多的解

Monte Carlo算法 (续)

■偏真算法



- 没有必要返回频数最高的解，一次true超过任何次数的false
- 重复调用k次一致、p正确、偏真的MC算法，可得到一个$(1-(1-p)^k)$正确的算法
  - 对于55%正确的偏真算法：重复调用4次可得到95%正确的算法，重复调用6次就可得到99%正确的算法，且$p > 1/2$的要求可放宽到$p > 0$

回顾抽奖问题

- 设抽奖箱中有不少于a $(0 < a < 1)$比例的一等奖，有放回的抽奖多少次可以保证抽到一等奖的概率不小于b $(0 < b < 1)$？
- 对立事件：
  至少抽到一次一等奖 vs. 抽到的全部不是一等奖
- 假设抽奖k次，至少抽到一次一等奖的概率
  $\geq 1 - (1 - a)^k \geq b$
  即$k \geq \log_{1-a}(1 - b)$

假设a=1%, b=0.9，则k≈230

Monte Carlo算法应用——主元素问题

■问题：设$A[1..n]$是含有n个元素的数组，若A中等于x的元素个数大于$n/2$，则称x是数组A的主元素

➢注：若存在，则只可能有1个主元素

■例：数组$A=\{3, 2, 3, 2, 3, 3, 5\}$，共7个元素，其中元素3出现4次，占一半以上，因此A存在主元素3

Monte Carlo算法应用——主元素问题(续)

```
MAJ(A)
1   i ← RANDOM(1, n)
2   x ← A[i]
3   k ← 0
4   for j ← 1 to n do
5       if A[j] = x
6           k ← k + 1
7   return (k > n/2)
```

- 返回true：A含有主元素x，算法一定正确
- 返回false：元素x不是A的主元素，算法可能错误（仅包含主元素时出错）
- A确实包含一个主元素时，x为主元素的概率大于1/2

MAJ是偏真1/2正确的算法

Monte Carlo算法应用——主元素问题(续)

■算法改进：通过重复调用技术降低错误概率

```
MAJ2(A)
1   if MAJ(A)
2       return true
3   else
4       return MAJ(A)
```

- MAJ2也是一个偏真算法
- A存在主元素时，MAJ2返回true的概率：
  - MAJ第一次返回true的概率 $p > 1/2$
  - MAJ第一次返回false且第二次返回true的概率 $p(1-p)$

  总概率：$p + p(1-p) = 1 - (1-p)^2 > 3/4$

MAJ2是偏真3/4正确的算法

Monte Carlo算法应用——主元素问题(续)

■算法改进：通过重复调用技术降低错误概率 (续)

- 重复调用MAJ的结果是相互独立的
- 当A含有主元素时，k次重复调用MAJ均返回false的概率为$(1-p)^k = 2^{-k}$
- 在k次调用中，只要有一次MAJ返回true，即可判定A有主元素
- 当需要控制算法出错概率小于$\varepsilon > 0$时，相应算法调用MAJ的次数为：

$$\varepsilon = 2^{-k} \Rightarrow k = \left\lceil \lg \frac{1}{\varepsilon} \right\rceil$$

时间复杂度为$O(n\lg(1/\varepsilon))$

```
MAJMC(A, ε)
1   k ← ⌈lg(1/ε)⌉
2   for i ← 1 to k do
3       if MAJ(A)
4           return true
5   return false
```

注意，这里只是用此问题来说明MC算法，实际上对于判定主元素问题存在$O(n)$的确定性算法

Monte Carlo算法应用——矩阵乘法验证

■问题：设**A, B, C**为$n \times n$矩阵，如何判定**AB=C**是否正确？

- ➢通过**A·B**的结果与**C**比较
  - 传统方法：$O(n^3)$
  - 当n非常大时，确定性算法：$\Omega(n^{2.37})$

■Monte Carlo算法

- ➢可在$O(n^2)$内解此问题，但存在一个很小的误差ε
- ➢设**X**是一个长度为n的0/1二值行向量，将判断**AB=C**改为判断**XAB=XC**

Monte Carlo算法应用——矩阵乘法验证(续)

■例：

$$\mathbf{A}=\begin{bmatrix}1&2&3\\4&5&6\\7&8&9\end{bmatrix}\qquad \mathbf{B}=\begin{bmatrix}3&1&4\\1&5&9\\2&6&5\end{bmatrix}\qquad \mathbf{C}=\begin{bmatrix}11&29&37\\29&65&91\\47&99&45\end{bmatrix}$$

- 设**X** = (1, 1, 0)
  - **XA** = (5, 7, 9) **A**矩阵1、2行求和
  - (**XA**)**B** = (40, 94, 128) **AB**矩阵1、2行求和
  - **XC** = (40, 94, 128) **C**矩阵1、2行求和
  - 返回true？ **错误！**
- **X** = (0, 1, 1)
  - **XA** = (11, 13, 15)
  - (**XA**)**B** = (76, 166, 236)
  - **XC** = (76, 164, 136)
  - 返回false **正确！**

Monte Carlo算法应用——矩阵乘法验证(续)

■Monte Carlo算法：设**X**是一个长度为n的0/1二值行向量，将判断**AB**=**C**改为判断**XAB**=**XC**

1. 计算$\mathbf{X}_{1\times n}\,\mathbf{A}_{n\times n}$ → n^2次数乘
2. 计算$(\mathbf{XA})_{1\times n}\,\mathbf{B}_{n\times n}$ → n^2次数乘
3. 计算$\mathbf{X}_{1\times n}\,\mathbf{C}_{n\times n}$ → n^2次数乘

```
GOODPRODUCT(A, B, C, n)
1   for i ← 1 to n do
2       X[i] ← RANDOM(0, 1)
3   if (XA)B = XC
4       return true
5   else return false
```

时间复杂度为$O(n^2)$

- 返回false：算法一定正确
- 返回true：仅对$\mathbf{X}_i$=1的对应行进行了求和验证，算法可能错误
- 若**A**·**B**与**C**的第i行不同且$\mathbf{X}_i$=0则出错，误判**AB**=**C**，出错概率不超过1/2

Monte Carlo算法应用——矩阵乘法验证(续)

■算法改进

```
REPEAT_GOODPRODUCT(A, B, C, n, ε)   //出错概率ε
1   k ← ⌈lg(1/ε)⌉
2   for i ← 1 to k do   //重复k次
3       if GOODPRODUCT(A, B, C, n) = false
4           return false
5   return true
```

时间复杂度为$O(n^2\lg(1/\varepsilon))$

- 出错概率 $\varepsilon \le 2^{-k}$
- REPEAT_GOODPRODUCT是偏假$(1-\varepsilon)$正确的算法